Composition and Verification of Web Service Based on Shared Resources

Yuejun Liu¹,²
¹School of Computer Engineering and Science, Shanghai University, Shanghai 200444, Shanghai, China
²School of Software Engineering, Anyang Normal University, Anyang 455002, Henan, China

Jing Su
School of Software Engineering, Anyang Normal University, Anyang 455002, Henan, China

Abstract
With the increase of Web services provided on the Internet, a larger granularity service which consists of many single function services or simple configuration services and meets individual user requirements is paid more and more attentions by researchers. To deal with modeling and verification of composition services based on shared resources, this paper presents resource transition system (RTS) and extends it to composition resource transition system (C_RTS). The C_RTS generation algorithm is given to construct the composition model which states number will expand rapidly. So, we describe how to reduce and verify the model based on the constraints between Web services. Then Probability Computing Tree Logical (PCTL) is introduced to better express the quantity probabilities. Finally, an example demonstrates the effectiveness of the method.

Key words: Web Service; Resource Transition System; Composition Model; Model Checker

1. INTRODUCTION

Service-Oriented Software (SOS) is the mainstream software form on the Internet, which shifts the traditional “terminal computing” to “web services”. Web service has encapsulation, loose coupling, cross-platform, dynamic composition and other features. Along with Web service standardization constantly improving, more and more service providers release their specific function service packs to the Internet (Jula et al., 2014). More and more researchers pay attention to how to compose single function Web services to a large granularity Web services for more complex requirements or individual requirements (Gold et al., 2004; Dustdar and Schreiner, 2005; Hatzi et al., 2012).

Because of distribution, dynamism and heterogeneity of the Internet, resources competition, service disruptions, communication failures and other anomalies occurred at any time will greatly affect the availability, and reliability of Web service composition (Shen et al., 2014). Therefore, we need to ensure the quality of composition service. Software verification is an important approach which can ensure software products correctness and improves software products reliability. At present, it is a mainstream verification technology in industry (Silvaet al., 2008)). The design and implementation of Service-Oriented Software product is different from traditional software product. They are usually composed of different components through standard interfaces and interaction, which are written by different program languages, distributing in the different network nodes and running on different platforms. So, the correct modeling and verification of combination services are the premise to ensure software availability.

The rest of the paper is organized as follows: Section 2 introduces the related work about service composition. Section 3 describes how to construct the composition model of Web Service. Section 4 discusses model property presentation. Section 5 gives an example to illustrate the process of service composition and verification, and finally, Section 6 concludes the study and comments on future work directions.

2. RELATED WORK

Many research institutions and scholars have studied various modeling and verification methods for service composition: (Fan et al., 2013) constructs the reliable service composition based on Petri net, which can verify the behaviors of service composition at designing and repairing process. (Du et al., 2014) presents a Petri net-based method to address message mismatches, state-space explosion and lacking of generating execution paths in Web service composition. (Ren and Liu, 2012) shows a service behavior automata to describe substitutability and verification of service composition. (Rai et al., 2015) proposes Web service composition algebra to capture the recursive nature in a Web service composition scenario. Moreover, studies also find many better methods to process service discovery: (Sirin et al., 2004; Kuzu and Cicekli, 2012; Hatzi et al., 2012) transform the Web service composition into a planning problem to discover atomic services. (Zhao et al., 2012; Zhang et al., 2010; Alrifai and...
Risse, 2010; Mallaya et al., 2015) respectively utilize Particle Swarm, Ant Colony, Skyline and user preference methods to discover atomic services and complete service composition. However, current researches on service composition mainly focus on how to compose services and how to select services from many services. Few people study service composition based on resource contention, this article proposes the composition resource transition system (C_RTS) and illustrates how to construct composition model, reduce model and verify model based on the competitive resources.

3. CONSTRUCT COMPOSITION MODEL OF WEB SERVICE

Formal verification is a mathematical method. Mathematical language is precise, consistency and suitable for logical reasoning, so the formal verification also can be proved and guaranteed (Newcombe et al., 2015). Formal verification divided into two forms: theorem proving and model checking. Because theorem proving cannot automate, it needs human involvement. Theorem proving requires the participants more domain knowledge. Model checking is self-executing and higher efficiency than theorem proving, but it needs to search the whole state-space and spend more time (Baier and Katoen, 2008).

Transition system is a model which describes the relation between states of the system and is usually an underlying digraph. Nodes of the graph represent states which describe the system behaviors at a special moment; edges of the graph represent the transitions behaviors between states. Transition system requires that each state must have at least a transition to other states, i.e., there is no deadlock state. To facilitate the process, if the system does really exist deadlock states, we can add a particular state s₀ and some transitions to resolve the deadlock problem. Each of the added transitions comes from one deadlock state and goes to the particular state s₀. For example, transition (s₀, s₁, s₂) represents that it comes from the deadlock state s and goes to the particular state s₀. At the same time, we also add a transition from state s₀ to itself.

In real life, however, the service depends on resources and the resources are usually shared. If the program resource as running condition has been changed, the running results are different. To solve this problem, we extend traditional transition system to the resource transition system by adding the resource and condition (Baier and Katoen, 2008).

**Definition 1.** Resource Transition System (RTS): A RTS can be defined as a tuple (Q, Act, R, VAR, →, q₀, g₀), where (Liu and Su, 2015)

- Q is a finite set of positions,
- Act is a finite set of actions,
- R is a finite set of resources,
- VAR: R × Act → R is a resource variable function. The resource R is affected by the action Act,
- → ∈ Q × Con R × Act × Q is a transition relation, which describes resource requirement of the transition,
- q₀ ∈ Q is a set of initial positions, and
- g₀ is a set of initial conditions.

Because of using the Resource Transition System (RTS), we can depend on the current specified resource to choose the appropriate transition path, i.e., selecting branch. That is to say, the execution of the service composition system is constrained by resource and conditions.

The traditional Transition System and the Resource Transition System all can construct the model of a separate service. If we need more complex functions or larger grained services, some atom services must be combined. When two or more atomic services are mutually independent, i.e., they don’t need to communicate each other and don’t need to compete for resources, they can be combined directly. Otherwise, when multiple services are dependent and interactive, it is more trouble to combine. Concurrency means two or more services together run in the same time period, which breaks the closure property of the service and shows the cross, non-blocking and non-reproducibility characteristics. Most often, concurrent services present mutual exclusion due to the competition for resources. To describe this problem we introduce the Composition Resource Transition System (C_RTS), which is an extended form of the Resource Transition System.

**Definition 2.** The Composition Resource Transition System (C_RTS): Let RTS₁ = (Q₁, Act₁, R, VAR₁, →₁, q₀₁, g₀₁) and RTS₂ = (Q₂, Act₂, R, VAR₂, →₂, q₀₂, g₀₂) (where i = 1, 2) is two RTS based on sharing resource R, then the composition resource transition system can be defined as:

C_RTS = (Q₁ × Q₂ × R, Act, R, VAR, →, q₀₁ × q₀₂ × x₀, g₀₁ × g₀₂), where

Q₁ × Q₂ × R is a set of positions with resource,

Act is a union set of Act₁ ∪ Act₂, the actions of Act₁ and Act₂ have no duplicate names,

VAR: when α ∈ Actᵢ, VAR(α, r) = VAR(α, rᵢ),

→: location < qᵢ₁, qᵢ₂, r > perform the action α to produce a new transition when the conditions Con(R) are satisfied: If α ∈ Act₁ and qᵢ₁ → qᵢ₁', then the composition transition is < qᵢ₁, qᵢ₂, r > → < qᵢ₁', qᵢ₂, VAR₁ (r) >; If α ∈ Act₂ and qᵢ₁ → qᵢ₂', then the composition transition is < qᵢ₁, qᵢ₂, r > → < qᵢ₂', VAR₂ (r) >,

q₀₁ × q₀₂ × x₀ is the initial position, and
The approach is to reduce those unreachability states and deadlock states. Moreover, the special formula $P_01$ can only be true or false, which represents the quality property.

\[ P_01 = \exists \phi \quad \text{where} \quad \phi \text{ is a PCTL path formula.} \]

The number of states will be extended to $|Q_1| \times |Q_2| \times |R|$ and the transition also rapidly increasing with the expansion of states. Therefore, we must reduce the number of states and transitions in the implementation process. The simplest and most effective approach is to reduce those unreachable states and deadlock states. We can search the directed graph in accordance with actions and transitions one by one from the initial state. All unreachable states should be reduced. At the same time, we also reduce the deadlock states, which can only accept transitions, from the transition system.

**Algorithm 1. C_RTS Generation Algorithm**

**Input:** $RTS_i = (Q_i, Act_i, R, VAR_i, -\tau, q_{i0}, g_{i0})$ (where $i=1, 2$)

**Output:** the composition model $C_RTS$ of $RTS_i$

**BEGIN**

DO WHILE \{< Q \times Q \times R \} is not empty

Get a new composition position $q_n = < q_1, q_2, r >$;

DO WHILE \{the action set of $q_1$ and $q_2$ in the position $q_n$\} is not empty

Get a new action $\alpha$;

IF $\alpha \in Act_1 \cap Con(r) \cap q_1 \rightarrow q_1'$

THEN $< q_1, q_2, r > \rightarrow < q_1', q_2, VAR_1 (r) >$;

ENDIF

IF $\alpha \in Act_2 \cap Con(r) \cap q_2 \rightarrow q_2'$

THEN $< q_1, q_2, r > \rightarrow < q_1, q_2', VAR_2 (r) >$;

ENDIF

\{ the action set of $q_1$ and $q_2$ in the position $q_n$ \} = \{the action set of $q_1$ and $q_2$ in the position $q_n$ \} - $\alpha$;

ENDDO

\{ $Q \times Q \times R$ \} = \{ $Q_1 \times Q_2 \times R$ \} - $q_n$

ENDDO

**ENDBEGIN**

According to the definition of C_RTS, we know the size of the composition model will rapidly expand. The number of states will be extended to $|Q_1| \times |Q_2| \times |R|$ and the transition also rapidly increasing with the expansion of states. Therefore, we must reduce the number of states and transitions in the implementation process. The simplest and most effective approach is to reduce those unreachable states and deadlock states. We can search the directed graph in accordance with actions and transitions one by one from the initial state. All unreachable states should be reduced. At the same time, we also reduce the deadlock states, which can only accept transitions, from the transition system.

**4. PROPERTIES VERIFICATION**

This article adopts the model checking method based on temporal logic: the system is modeled as a Composition Resource Transition System and the user properties are expressed as temporal logic formulas. Then, we travel all the states of the state space and verify whether the model is satisfied with the properties by means of model checking tools. Because the state number of system model is finite, the travel always can be ended.

As an important branching temporal logic, Computing Tree Logic (CTL) can better express the consistency and unambiguity of system property formulas. Probability Computation Tree Logic (PCTL) is the extend form of Computing Tree Logic (CTL), which can quantitatively describe probability(Baier and Katoen, 2008).

**Definition 3. Probability Computation Tree Logic (PCTL) formula:**

“True”, “False” and atomic proposition variable are PCTL state formulas,

If $\phi_1, \phi_2$ are PCTL state formulas individually, $X \phi_1, \phi_1 U \phi_2$ and $\phi_1 U \phi_2$ are PCTL path formulas,

If $\phi_1, \phi_2$ are PCTL formulas, $\neg \phi_1$ and $\phi_1 \land \phi_2$ are also PCTL state formulas, and

If $\phi$ is path formula, $P_{\phi}(\phi)$ is PCTL state formulas.

Where, $m$ is an integer greater than or equal to 1, which represents step length. ~$J$ represents a probability bound of $\phi$. When $J=1$ or $J=0$, $P_{\phi}(\phi)$ represents that $\phi$ is true or false, which represents the quality property.

Moreover, the special formula $P_{\min} = \exists \phi (\phi)$ (or $P_{\max} = \exists \phi (\phi)$) represents the maximum (or minimum) probability when $\phi$ holds(Liu and Su, 2016).

PRISM developed by M.Kwiatkowska is a common model checking tool, which supports LTL, CTL and PCTL, etc. Once have the model and properties been input to PRISM, it will return whether the model satisfies the properties. If the result is false, it shows that the constructed model is not satisfied with user properties. Then, we need modify the model until the user’s properties are satisfied. Generally, reachability, safety, liveness and fairness are four basic properties to be verified.
5. EXPERIMENTS

Assume that there are two online services: financial reimbursement service A and financial remittance service B. For simplicity, assume that transact business window is only one R, that is to say, the number of public resources is one. The resource can only be used by one service exclusively and not be shared by multiple services at one time. According to the description above, the following is their composition and verification process.

(1) First, we construct the transition system model for each service (Figure 1). Each service is divided into three parts: give identifier (i), apply for service (a), and financial reimbursement (remittance) (m). i-A, a-A and m-A are respectively giving identifier, applying for service and financial reimbursement (remittance) section of service A, which i-A is the initial state of the service A. Service B is the same as A.

![Figure 1. State transition of A and B services](image)

(2) The transition execution is bound by special resource. In order to better describe the system model, we extend the transition system to the Resource Transition System RTS (Figure 2). For simplicity, let resource \( r = 1 \) which indicates the service window is idle, i.e., the resource can be used freely; \( r = 0 \) presents the service window is busy, i.e., the resource is being used exclusively and the transition cannot do anything but wait. Only is the service window idle, a-A or a-B state can apply to enter into state m-A or m-B. When the financial reimbursement or remittance business is be done, the resource \( r \) is released, i.e., \( r = r + 1 \).

![Figure 2. RTS of A and B services](image)

(3) According to C_RTS algorithm, the C_RTS model is given as shown in Figure 3. The total number of states is \( |Q_1| \times |Q_2| \times |R| = 3 \times 3 \times 2 = 18 \). The state \( <i-A, i-B, r=1> \) and \( <i-A, i-B, r=0> \) are initial states.
(4) Reduce the C_RTS model. Reduction can be divided into two steps: first, starting from initial states we travel the digraph and reduce those unreachable states according to the reachability. That is to say, we obtain a reachable state model C_RTS through the reduction process. The results are shown in Figure 4.

Then, we eliminate the deadlock states and get the reduction system model shown in Figure 5.
(5) Finally, we get the C_RTS system model shown as Figure 6, which consists of eight states and each state has different meaning defined as follows in Table 1.

<table>
<thead>
<tr>
<th>States</th>
<th>Represent Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt; i-A, i-B, r=1&gt;</td>
<td>A and B possess identifier; resource r is free.</td>
</tr>
<tr>
<td>1</td>
<td>&lt; i-A, a-B, r=1&gt;</td>
<td>A possesses identifier; B tries to apply for service; resource r is free.</td>
</tr>
<tr>
<td>2</td>
<td>&lt; a-A, i-B, r=1&gt;</td>
<td>A tries to apply for service; B possesses identifier; resource r is free.</td>
</tr>
<tr>
<td>3</td>
<td>&lt; a-A, a-B, r=1&gt;</td>
<td>A and B all try to apply for service; resource r is free.</td>
</tr>
<tr>
<td>4</td>
<td>&lt; i-A, m-B, r=0&gt;</td>
<td>A possesses identifier; B processes remittance; resource r is busy.</td>
</tr>
<tr>
<td>5</td>
<td>&lt; m-A, i-B, r=0&gt;</td>
<td>A processes reimbursement; B possesses identifier; resource r is busy.</td>
</tr>
<tr>
<td>6</td>
<td>&lt; a-A, m-B, r=0&gt;</td>
<td>A tries to apply for service; B processes remittance; resource r is busy.</td>
</tr>
<tr>
<td>7</td>
<td>&lt; m-A, a-B, r=0&gt;</td>
<td>A processes reimbursement; B tries to apply for service; resource r is busy.</td>
</tr>
</tbody>
</table>

(6) The execution environment is a common machine with Core i3-2130.4GHz and 4Gb RAM. The operating system is windows 8, 64bit and PRISM is 4.2 Beta. We assume the probabilities of service A and B are 0.6 and 0.4 respectively. Then, according to the Composition Resource Transition System shown in Figure 6, we write out the corresponding program:

```mdp
module fs_ab
s:[0..7] init 0; // Initialization
```
Next, we respectively study the liveness and safety property to show the effectiveness of the method. 

Liveness: we detect how the probabilities of $P_{\text{min}}=?\left[ \text{true} \U 0.4 \left( s=4 \right) \mid \left( s=7 \right) \right]$ change. Figure 7 represents the minimal probabilities increases accompany with the increasing of the steps, where the reimbursement service process A and the remittance service process B are executed separately. After 20 steps, we can say the property formula has almost 100% opportunities to occur. This result is also can be seen from $P_{\text{min}}=?\left[ \text{true} \U 0.4 \left( s=4 \right) \mid \left( s=7 \right) \right]$, i.e., this event that service A or service B be executed must occur.

![Figure 7. Steps and probabilities](image)

Safety: $P_{\text{max}}=?\left[ \text{G} \ F \ s=4 \ & \ s=7 \right]$ represents the maximum probability which the reimbursement service process A and the remittance service process B are executed concurrently. By PRISM model checker weget the execute result 0 which indicates that service A and service B cannot occur simultaneously at any time. This result meets the safety property.

6. CONCLUSIONS

In order to guarantee the correctness of a larger grained service composed of simple services, this article proposes service composition and verification method based on shared resource. The method gives model composition and model reduction through the Composition Resource Transition System based on the constraints between services and resources. Finally, an example demonstrates the effectiveness of the method. In the future, we will introduce “rewards” to further research the functional and non-functional properties of composition service model.

Acknowledgements

This work was partly supported by Soft Science Projects of Henan Province (132400410249), Natural Science Research Program of the Department of Education of Henan Province (12A520001).

REFERENCES


